Document

D6.3 - Evaluation and Guidelines for Developing POPEYE Applications

Project Number:	IST-2006-034241
Project Title:	POPEYE
Document Type:	Deliverable

Planned Completion Date:	30.04.2008
Actual Completion Date:	30.10.2008
Editors:	Christian Melchiorre (SOFTECO)
Participants:	Refer to contributors list
Workpackage:	WP6
Nature:	Report
Dissemination Level:	Public
Version:	2.0
Total Number of Pages:	39
File name:	POPEYE_D6.3-V2_081030_PU.doc

Abstract

This deliverable document D6.3 entitled "Evaluation and Guidelines for developing POPEYE Applications" provides;

- the description of POPEYE proof-of-concept application, to be used as a sort of user manual for the POPEYE environment,
- guidelines for the evaluation of POPEYE proof-of-concept application and detailed instructions for the extension of the application itself,
- instructions for developers on how to develop mobile P2P e-Collaboration application on top of POPEYE architecture, describing how to exploit POPEYE Plug-in architecture to create new applications by adding new functionality.
- all source-code documentation for building POPEYE applications

Keywords List

Proof-of-concept application evaluation guidelines, User Manual, Plug-in, Extension

Executive Summary

This document contains Deliverable D6.3 – Evaluation and Guidelines for Developing POPEYE Applications, due in month M24.

The document is composed by three main parts:

The first part includes a description of POPEYE proof-of-concept application as it is at the end of the project, including the Plug-ins developed during the course of the project. This part is meant to be used as a sort of user manual for the POPEYE environment.

The second part of the deliverable contains guidelines for the evaluation of POPEYE proof-of-concept application and detailed instructions for the extension of the application itself, through the mechanism made available by the Plug-in architecture. This part also contains a description of the Network Topology Configuration Tool, an application developed internally by GET-ENST that was used for evaluation and testing of POPEYE.

The third and last part of the deliverable provides instructions for developers on how to develop mobile P2P e-Collaboration application on top of POPEYE framework, describing how to exploit POPEYE Plug-in architecture to create new applications by adding new functionality.

We hope the present public guidelines and instructions effectively facilitates the development of mobile P2P e-Collaboration application on top of POPEYE framework. Further, developers are invited to refer to the POPEYE web site at www.ist-popeye.eu for more publicly available documentation, and may even join the POPEYE user community.

Revision history

Version	Date	Description, Editors
1.0	21.03.2008	Initial version, Christian Melchiorre
2.0	30.10.2008	Updated version to incorporate the elements required after the Final Review as follows: ▪ providing all source-code documentation for building POPEYE applications (Annex A) Christian Melchiorre.

Contributors

Name	Last Name	Company	Email
Christian	Melchiorre	SOFTECO	christian.melchiorre@softeco.it
Heiner	Bunjes	OFFIS	heiner.bunjes@offis.de
Daniel	Wichmann	OFFIS	daniel.wichmann@offis.de
Francisco	Bas Esparza	GET-ENST	bas@enst.fr
Marcel	Arrufat	URV	marcel.arrufat@urv.cat
Gerard	París	URV	gerard.paris@urv.cat
Patrizio	Pelliccione	UDA	pellicci@di.univaq.it
Nicolas	Berthet	THC	nicolas.berthet@fr.thalesgroup.com

Acronyms

Acronym	Meaning
CoMa	Collaborative Map
CWE	Collaborative Working Environment
LAN	Local Area Network
MANET	Mobile ad-hoc network
P2P	Peer-to-peer
PDA	Personal digital assistant
POI	Point Of Interest
WP	Workpackage

The present document has been produced in consistence with the definition of terms described in the POPEYE Glossary v1.0 accessible on the POPEYE web site http://www.ist-popeye.eu/

Table of Contents

Executive Summary **2**

Revision history **3**

Contributors **4**

Acronyms **5**

Table of Contents **6**

1 Introduction **7**

2 Short description of the evaluated system **8**

2.1 The POPEYE Application 9

2.2 The POPEYE Plug-ins 10

2.2.1 POPEYE Shared Spaces and the File Sharing Plug-in *11*

2.2.2 POPEYE Messaging and the Messaging Plug-in *14*

2.2.3 The Shared Presentation Plug-in *16*

2.2.4 The Voting Plug-in *20*

2.2.5 The Forum Plug-in *20*

2.2.6 The Anagram Game Plug-in *21*

2.2.7 The Collaborative Map Plug-in *24*

3 Evaluation of the POPEYE Application **25**

3.1 Internal evaluation 25

3.2 Evaluation by demonstration events 25

3.3 The Network Configuration Utility as an evaluation tool 27

3.3.1 Motivation *27*

3.3.2 Functionality *28*

3.3.3 How it works *28*

3.3.4 User Manual *30*

4 POPEYE Application Development Guidelines **32**

4.1 The POPEYE Plug-in based Framework 32

4.2 Developing new Plug-ins 33

4.3 Installing new Plug-ins 37

5 Conclusions **38**

ANNEX A: Popeye Source Code **39**

1 Introduction

This document is deliverable D6.3 – Evaluation and Guidelines for Developing POPEYE Applications. As result of task T6.3, which provides the summary and synthesis of the working activities and associated results achieved within WP6, this document provides guidelines for developing mobile P2P e-collaboration applications on top based on the enabling POPEYE architecture by exploiting the POPEYE Plug-in based framework.

The document begins with a description of the proof-of-concept application as it is at the end of the project, including the Plug-ins developed during the course of the project.

It then continues with a section dedicated to the process of evaluation of POPEYE proof-of-concept application, with reference to the feedback and experience gained from the two demonstration events that have occurred during the project period. This part of the document also contains a description of the Network Topology Configuration Tool, an application developed internally by GET-ENST that was used for evaluation and testing of POPEYE.

The final part of the document provides instructions for developers on how to develop mobile P2P e-Collaboration application on top of POPEYE framework, describing how to exploit POPEYE Plug-in architecture to create new applications by adding new functionality.

2 Short description of the evaluated system

This section deals with the description of the POPEYE proof-of-concept application as it is currently at the final stages of the project's duration. The usage of the application and its components (Plug-ins) already available are described.

The POPEYE system is intended to be used in the easiest way possible from the user's point of view. When POPEYE users want to collaborate they just have to start the POPEYE Environment on their wireless device and login to the POPEYE network. The device connects to a P2P overlay network which is built upon a mobile ad-hoc network (MANET).

In the next step a user can search for and join existing Workspaces or she/he creates a new Workspace and invites other available users to join him for collaboration. In POPEYE the Workspace is the term used to designate a group of users and the data and applications they share. The users who joined a Workspace form the Group that belongs to the Workspace. Sharing of data between all members of the Group is supported by the Shared Space, which is associated to each Workspace. The applications the users employ for collaboration (e.g., file sharing, group calendar, forum,...) can be plugged into the user's local POPEYE environment at runtime. We call those applications Plug-ins and their instances associated to a Workspace and having a specific configuration are named (Plug-in-) sessions. The POPEYE system is secure and context aware at all times.

The first implementation of POPEYE available as proof-of-concept application has been used as the basis of two demonstration events. The first one of these events took place in Paris on Oct. 26th, 2007 and was open to a restricted group of users (the POPEYE User Group) with the goal of testing and validating the POPEYE tool and architecture in a realistic situation.

The second event during which POPEYE was demonstrated was a more open, public showcase of POPEYE technologies set up in the context of a project open Workshop organised in conjunction with the MiNEMA event held in Glasgow on April, 1th 2008. The goal of the second demonstration of the POPEYE system was also to disseminate the project results to the wider IST, scientific and technical communities.

The results of both the demonstration events are collected into two proceedings documents publicly available[1].

[1] Publicly available POPEYE documentation can be obtained at www.ist-popeye.eu

2.1 The POPEYE Application

The current section of this deliverable describes the usage of the POPEYE tool as available at the end of the project. The following sections will illustrate the most relevant plug-ins already available at the moment of the issue of this document.

As soon as the user clicks on the POPEYE client icon on his laptop desktop, a start-up console window is shown asking for login information (username and password) as depicted in the following picture.

Fig. 1 - POPEYE login console

In this console registered users can enter otherwise other users can make the registration (through the "*new*" button). After making the registration, entering required information and making use of the certificate received into the desk, the user will see a console similar to the one in the next Figure.

Fig. 2 - POPEYE console

This is a sort of top-level menu for the POPEYE system. The round icons give access to the main commands available, which are described in the following text. The console shows some basic information about the user that has logged in to the system through this POPEYE instance. A small icon shows the current state of the connection. A part of the console toolbar is also used to contain some visual notification mechanisms (not apparent in the figure).

The first icon, starting from left, is used to open or create a Workspace. Upon clicking, a list of existing workspaces is displayed, and also the chance to define a new Workspace is given. Once an existing or a new Workspace is selected, the main window is displayed and the Workspace is opened.
The second icon is used to edit the user's preferences. A window with user's profile information is displayed and the user has the chance to edit it.
The third icon is for general application settings and preferences.
The fourth icon is search in the profiles. It opens a window where querying the context is possible and displays then the results of the search. From the result list it is possible to perform some operations, such as inviting a user to a Workspace.
The fifth icon is "Logout".

2.2 The POPEYE Plug-ins

The power and flexibility of the POPEYE software lies in the fact that its Plug-in based architecture allows easy extensibility of its functionality depending on the particular needs and requirement of each specific implemented application.

Plug-ins represent "*pieces*" of applications, which are used for collaboration, encapsulating specific functionality. They can easily be plugged into the local POPEYE System at runtime. Plug-ins can register themselves or can be retrieved by using a central software component in the POPEYE software, the so-called Framework Manager, which acts as an entry point to the POPEYE platform.

Multiple instances of the same Plug-in can run on the local POPEYE System (even in the same Workspace). Each of those instances represents an own Session with an individual configuration. Possible dependencies between plug-ins (a plug-in could need other plug-ins to be already loaded) are managed by the POPEYE plug-in infrastructure, a framework and tool for software architecture analysis.

The POPEYE plug-in infrastructure allows plug-ins to communicate each other; for instance it is possible to implement a map plug-in showing peers geographically dislocated in different rooms of the conference and it is possible interacting on the map to directly open a chat with one of the peers.

Extension points technology provided by the plug-in infrastructure permits to implement a plug-in that extends other existing plug-ins adding new functionalities (e.g., a new button in a plug-in toolbar with a new associated functionality, or a view window in a map plug-in - a map that could be too big to be readable when displayed in its entirety - giving an overview of the overall map and showing which part of the map is currently displayed in the map plug-in).

Plug-ins access the middleware services through the Plug-in API.

From the POPEYE client, sessions of Plug-ins can be executed in the context of a Workspace from inside the Workspace Explorer. Separate windows for each Plug-in sessions will be opened in a MDI (*Multiple Document Interface*)-like central window area, where they can be overlapped, minimised or maximised for easier user interaction.

Several Plug-ins have already been developed and included in the distributed POPEYE software prototype. These have been chosen selecting general purposes Plug-ins and Plug-ins that have their specific use in relation of the selected baseline scenario addressing the case of a symposium or conference organisation.

Each of the plug-ins currently available is described in one of the following subsections.

2.2.1 POPEYE Shared Spaces and the File Sharing Plug-in

Overview

One of the core services of the POPEYE framework is the "Data Management and Sharing Services Module" (WP5). This module provides a virtual common space for each of the workspaces where users (by means of plug-ins) and other internal modules can share files and objects.

The contents of each of these Shared Spaces are distributed within the different devices in the network so single peers do not need to maintain locally a copy of all the shared items. However, all the users taking part in the same Shared Space, maintain a global view and do not need to know the physical location of the files. It is duty of the "Data Management and Sharing Services Module" to decide where to place the shared data and where to locate them when they are requested.
The contents of the Shared Spaces follow a tree structure so that users can create directories and sub-directories to better organise their files.

In addition, the system stores certain attributes (called metadata) associated to the data. While some of these attributes can be modified by the users (e.g. author, description), some others are set by the system (e.g. creation date). In either case, users can use them to perform queries which allow them to identify documents they are interested in. These search mechanisms are provided by the "Data Search Mechanisms" (WP4).

The File Sharing Plug-in is a graphical application providing access to all the services provided by the "Data Management and Sharing Services Module" and the "Data Search Mechanism".

User Interface

Fig. 3 shows an example of Shared Space name « TEST » where users have created some folders and shared some files.

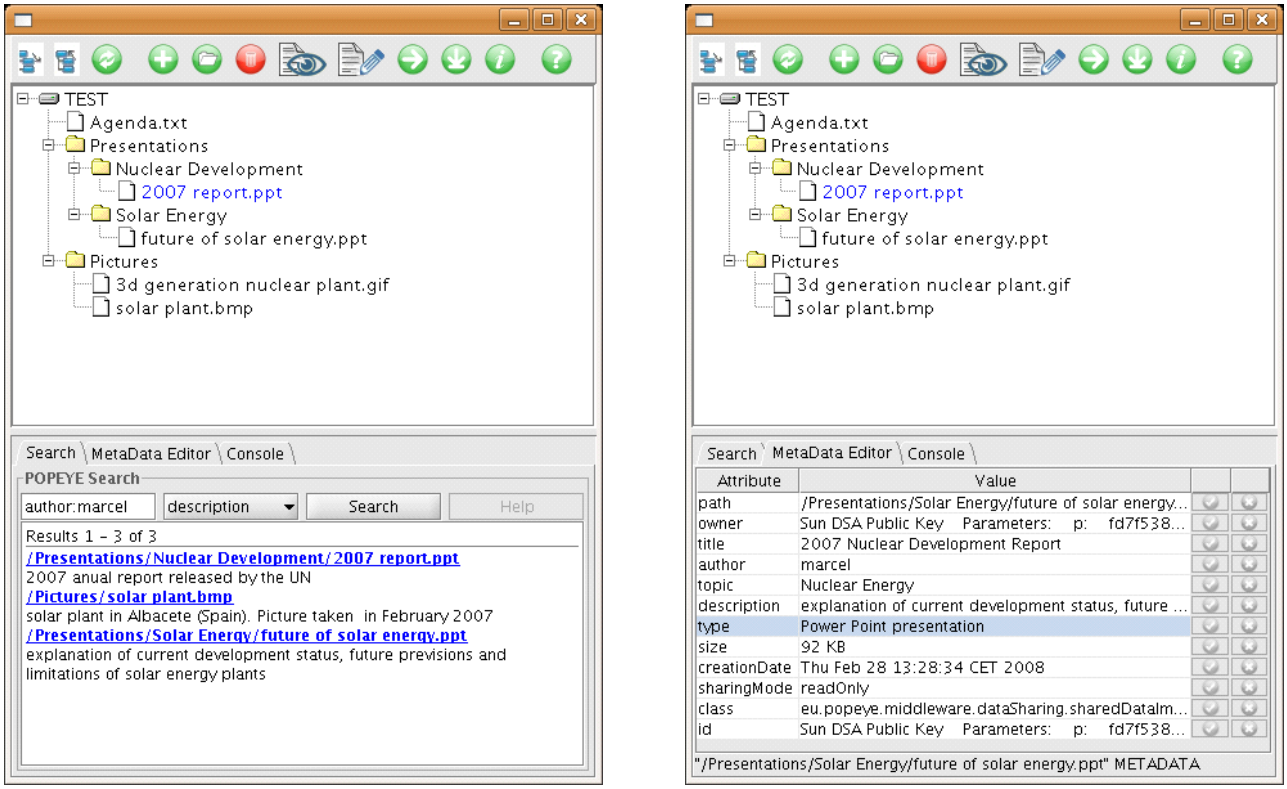

Fig. 3 - File Sharing Plug-in

The use of the plug-is is rather intuitive. The graphic interface is divided into three different parts:

- The ***toolbar*** contains a set of buttons in order to access the different functionalities (some of these functions together with some extra functionality can be accessed from the menu which appears when user presses the right button of the mouse over any node).:
  - "Expand All": expands all the nodes of the tree structure
  - "Collapse All": collapses all the nodes of the tree structure
  - "Refresh Tree": although automatic refresh is triggered by notifications of tree structure changes, we have experienced some problems. By clicking this button, we force an update
  - "Share Data": shares a new document in the shared space
  - "Create Directory": creates a new directory in the shared space
  - "Delete": removes a file or directory from the shared space. In case of directory, it must be empty before it is erased
  - "Access Read": opens the file in read only mode
  - "Access Write": opens the file in write mode
  - "Commit": commit the modifications (and Access Write must have been previously done)

-  "Download": Saves the whole or part of the documents in the Shared Space in a local directory so they are available even when disconnected from the network
-  "Get Metadata": retrieves the metadata associated to the data and displays it in the metadata panel
-  "About": shows information about the plug-in (author, data, version,...)

- The ***top panel*** shows the tree structure of the Sharing Space which name is indicated at the root node.
- The ***bottom panel*** is composed by three tabs and only one is visible at a time:
  - The **Search** tab contains a field used by users to specify their query parameters . For example, in Fig. 3, user searched for all the documents which *author* is *Marcel.* He also specified that he wanted the *description* of the matching documents to be displayed. In this case, three documents matched the used query.
  - The **Metadata editor** allows the display and modification of the attributes associated to the files (see Fig. 3).
  - The **Console** tab is used only for debugging purposes.

2.2.2 POPEYE Messaging and the Messaging Plug-in

Overview

The Popeye Messaging Plug-in allows all members inside a group to communicate by exchanging group and private messages. The plug-in graphical interface is similar to a chat room. In first place, all the members of the group who have launched the plug-in are listed in the membership list. Besides, messages sent to the scope of the whole group (i.e. sent to all the members of the group which launched the plug-in) are shown in the main board, common to all the participants. Furthermore, by selecting one participant from the membership list, it is also possible to send a private message, so only the sender and the receiver can follow (see) the conversation.

Technical considerations

The Popeye Messaging Plug-in demonstrates basic functionalities provided by communication services.
In first place, in order to achieve group communication, the plug-in creates a new named communication channel. Only the members of the group which created this named communication channel will receive messages from the channel. Thus, two of the most important functionalities of the channel are used:

- sendGroup: the message is received by all members.
- send: the message is received only by one member.

On the other hand, messages are handled by each node considering whether they are group or private messages, so they are shown in the main board or in a separate window, respectively. Besides, each member advertises itself when entering the chat room, so the other members can update the membership list.

Apart from this, the messaging plug-in also allows encrypted communication, so confidentiality is assured for plug-in messages. If encryption is needed, the plug-in will retrieve the workspace-shared key to encrypt the messages. Therefore, no person outside the workspace or the Popeye application will be able to intercept messages and process them.

User interface

Once the plug-in is launched, a window displays the different parts of the plug-in information. On the left side, the member list displays all the members that have launched the messaging plug-in. On the right side, a message panel shows all messages that have been sent to the chat room. Messages and event notifications will be shown in this panel, so that all members that launched the plug-in see the same information.

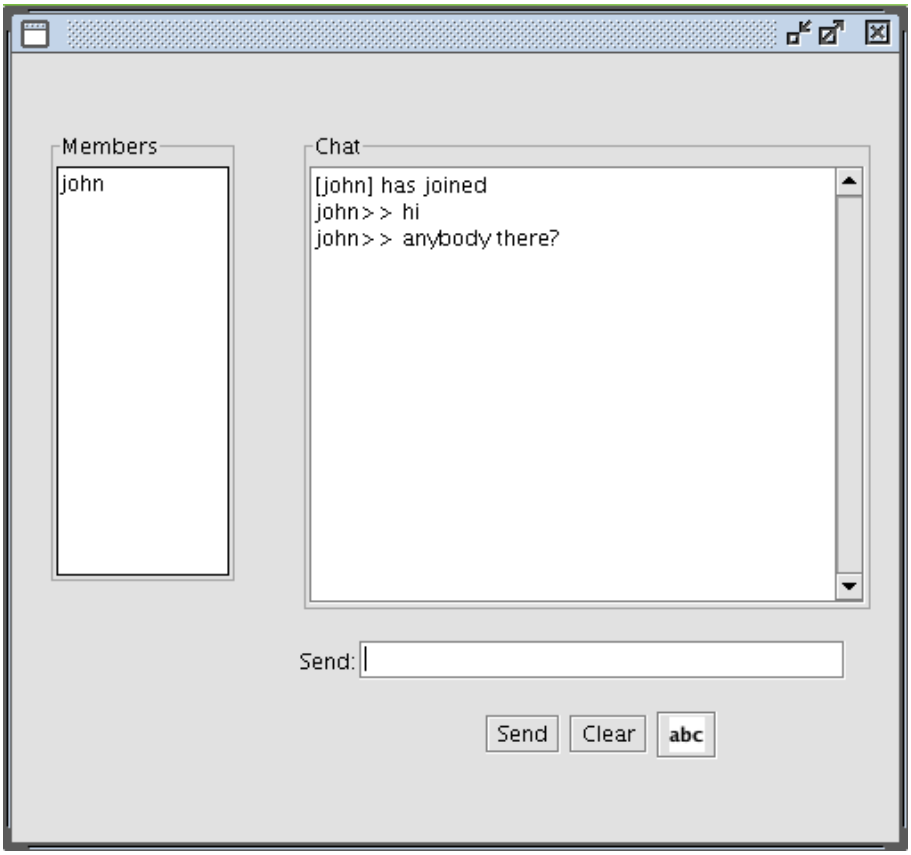

Fig. 4 - Messaging Plug-in

On the lower side of the plug-in, users can write text and send it to the chat room by pressing the send button (or the return key, as it is performed usually). Furthermore they can enable message encryption by clicking on the “abc” button. This button shows the status of the message delivery, that is, “abc” means that the text will be sent as clear text, and the button will contain a “lock” picture when the text to be sent is encrypted.

2.2.3 The Shared Presentation Plug-in

Overview

Within collaborative environments, it is very common to perform talks and presentations using a set of slides in order to reinforce the information given by the speaker. To do that, a slide projector would be necessary. This fact collides strongly with the spirit of POPEYE, where spontaneous networks can be created without any previously established infrastructure, not even a slide projector. To be even more strict, imagine the situation where an event is held in the open air: we wouldn't even have a wall where slides could be projected!
Taking these facts into account, we considered very convenient the implementation of a plug-in allowing the management and visualisation of presentations. We called it PDFViewer, due to the format of the supported files. In brief, with this application, the speaker controls the visualisation of his presentation on the audience devices, i.e. he/she decides which file is to be presented and the page number to be displayed at any moment. The audience devices are merely passive and follow all the orders received from the coordinator.

Technical considerations

This plug-in makes uses of the following Popeye modules:

- **Naming Service** is used to announce the name of the presentation and the identity of its coordinator
- **Data Management and Sharing Services** are used to share the displayed presentation so it is available to all the devices taking part.
- **Communication Services** are used to communicate the changes in the page number being currently displayed.

User Interface

Once the plug-in is launched, an initial window is displayed and the user is asked to choose one of two different operation modes. If the user selects *coordinator mode*, he/she will manage a presentation and will be able to select at any time, the presentation to be visualised and the page number to be displayed. Otherwise, if the user selects *presentation viewer mode, he/she* will be passively taking part in the presentation and will follow the coordinator actions.
Since more than one presentation can be held at the same time within the same workspace, the initial window displays a list containing the different presentations and their respective coordinators so the viewers can choose the presentation the want to join.

Fig. 5 shows an example of the plug-in initial window. In this case, only one presentation has been started: “presentation_name.pdf”, being “coordinator-name” its coordinator.

Fig. 5 - PDF Viewer initial window

Fig. 6 shows the coordinator window. It contains a File menu which allows the coordinator the selection of the file to be displayed. On the bottom, he/she can choose the page number to be displayed.

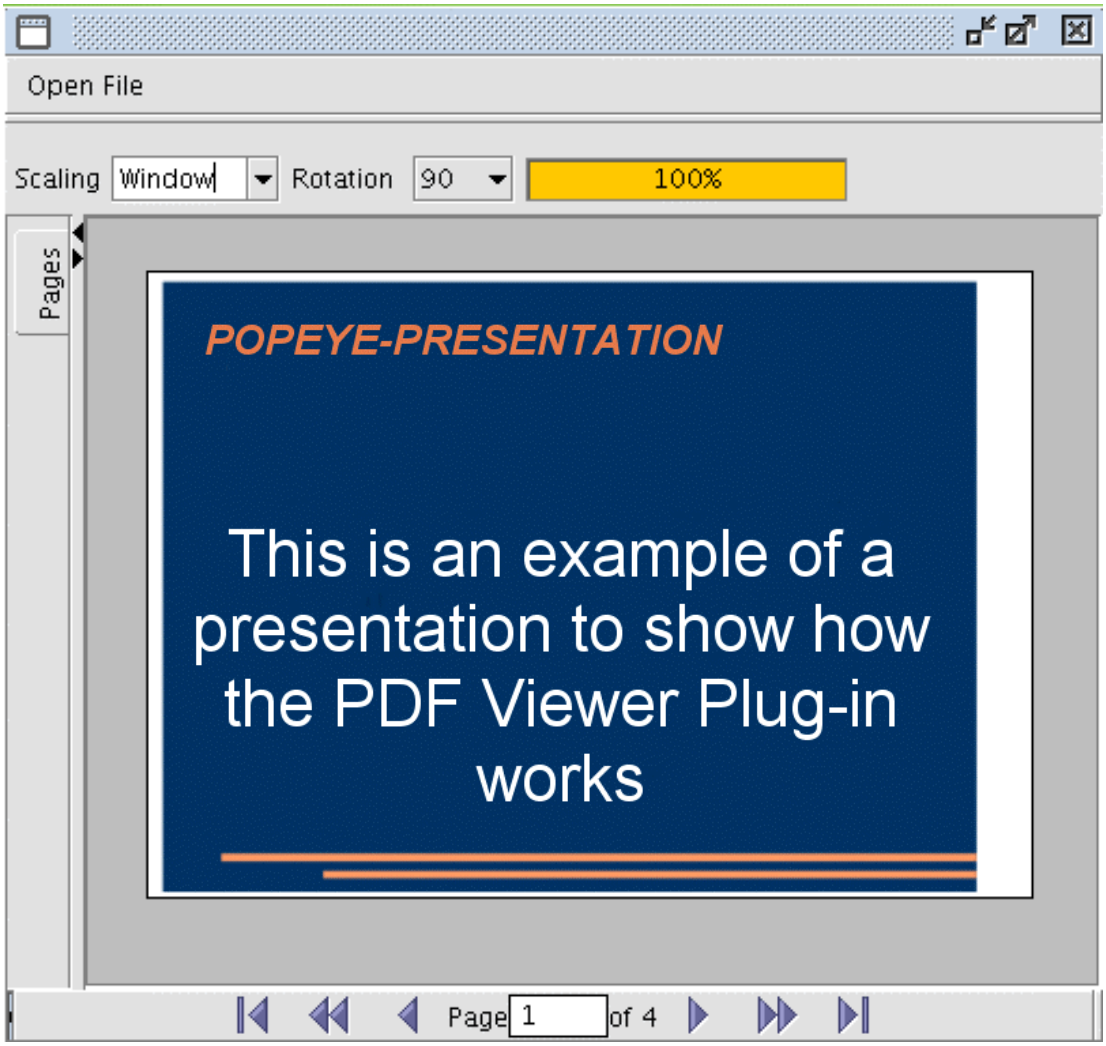

Fig. 6 - PDF Viewer: coordination mode

Fig. 7 shows the same presentation displayed in viewer mode. In this case, the user cannot interact with the presentation, therefore, no File menu or buttons allowing page changes are visible.

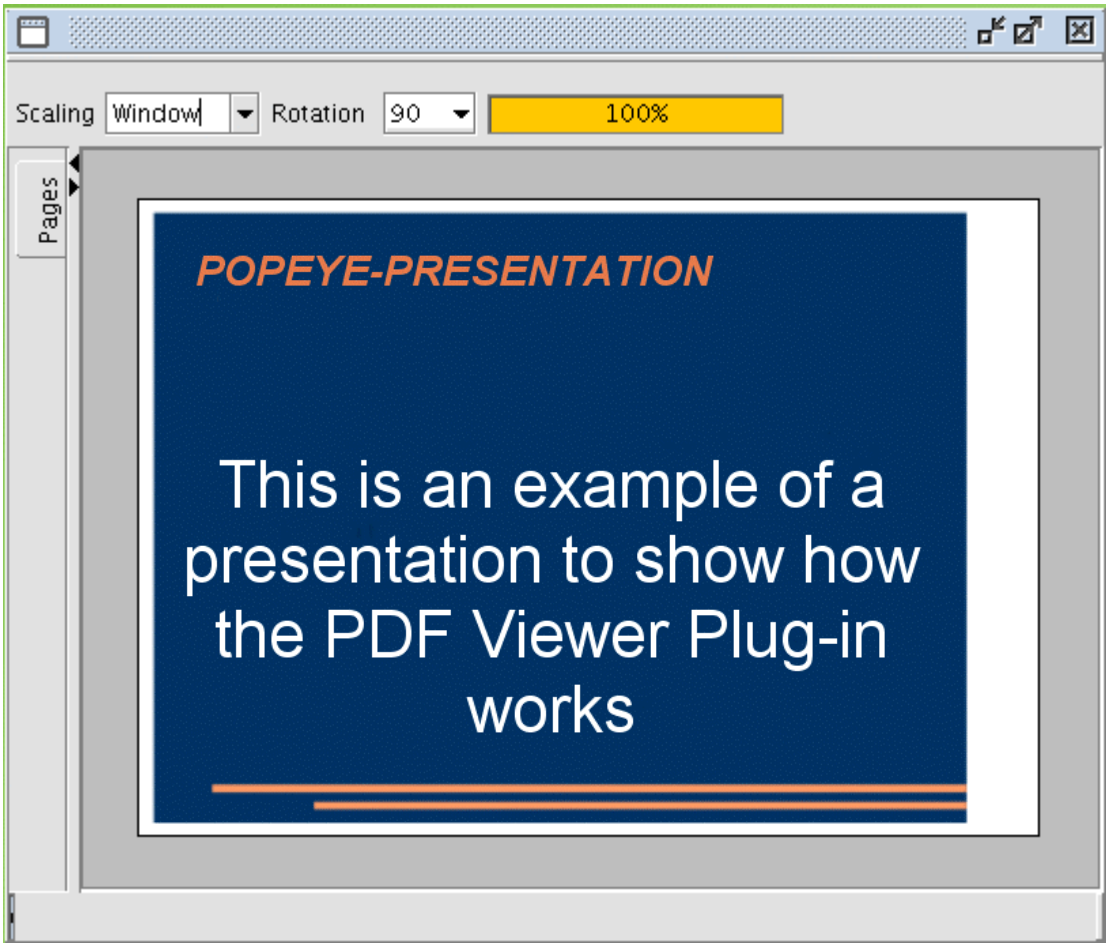

Fig. 7 - PDF Viewer: viewer mode

2.2.4 The Voting Plug-in

Overview

The voting plug-in implements a simple voting protocol to determine the preference of members of a Workspace on some subject. It allows any member to define a poll, with a question and some options. All the other members can then either choose one of the given options or "dismiss the issue", signalling they don't care. Results and some statistics are collected by the proponent of the poll, who can then publish to all the other members aggregated results. The proponent can also choose at one point to stop the poll, meaning no more answers will be accepted.

Technical Considerations

The voting plug-in makes use of both group multicast messages and unicast messages in the various stages of the poll. In detail, the poll proposal and results publishing is done in multicast by the poll proponent, while single preferences are sent in unicast mode to the proponent. This allows for some weak anonymity scheme, as only aggregated results are spread around. However, this plug-in has only demonstrational purposes and is by no means adequate to real voting.

2.2.5 The Forum Plug-in

Overview

The forum plug-in is a means of sharing information and opinions in a threaded, consistent way such as it is done on a Web forum. A workspace member can browse in the hierarchical data, write, read and reply in a threaded way. This plug-in makes use of data sharing, keeping thus persistence of data within the workspace lifespan.

2.2.6 The Anagram Game Plug-in

Overview

The Popeye Anagram Plug-in allows all members inside a group to play a game exchanging group messages. The game consists in creating the longest possible anagram with the given letters. The anagram is the result of rearranging the letters of a given word to produce a new word, using all the original letters exactly once. In this case, the user has 8 shuffled letters each round, so that he can form at least one 8-letter word during a certain period of time. The user will get points depending on the length of the word, in a way that a longer word will give the user more points.

Technical considerations

The Popeye Anagram Plug-in demonstrates basic functionalities provided by communication services.
In first place, in order to achieve group communication, the plug-in creates a new named communication channel. Only the members of the group which created this named communication channel will receive messages from the channel. In this case, the functionality used from the channel is:

- **sendGroup**: the message is received by all members.

User interface

The plug-in window is composed of different parts. On the upper left side the shuffled letters are shown to the user, so he can rearrange them and form new words. On the lower left side the user can verify whether the introduced word is valid or not (by clicking the “Check” button). If the word is a well-formed and existing word, it will be shown in green under the shuffled letters. Otherwise the word will appear in red.

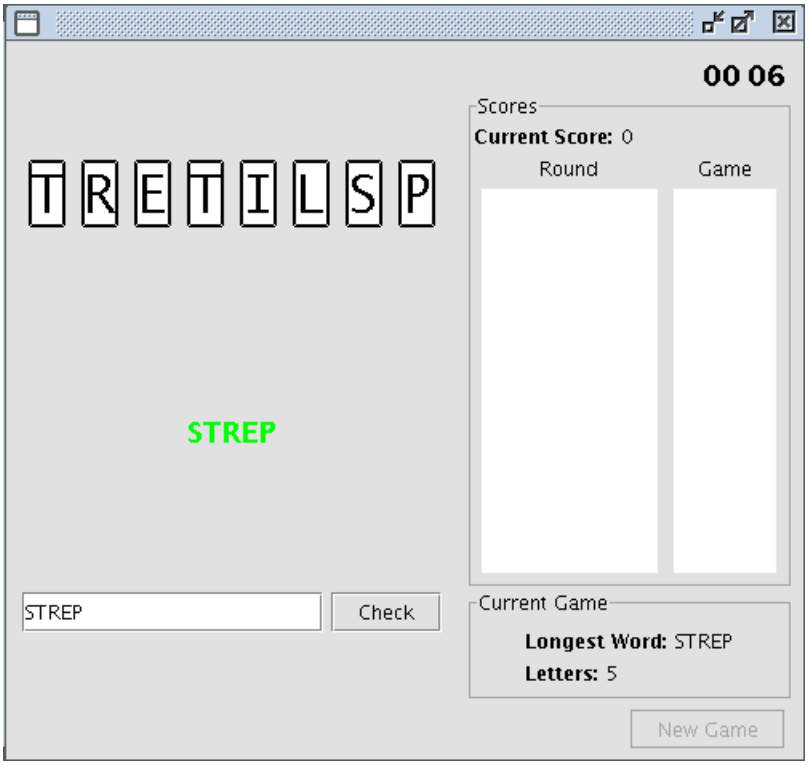

Fig. 8 - Anagram Plug-in game

On the right side, there are two panels with information about the game scores and the current game. On the upper right side, first there is information about the last played round. There is a list with the words constructed by all the players in the game. Next to it, there is the accumulated score for each player in the game.

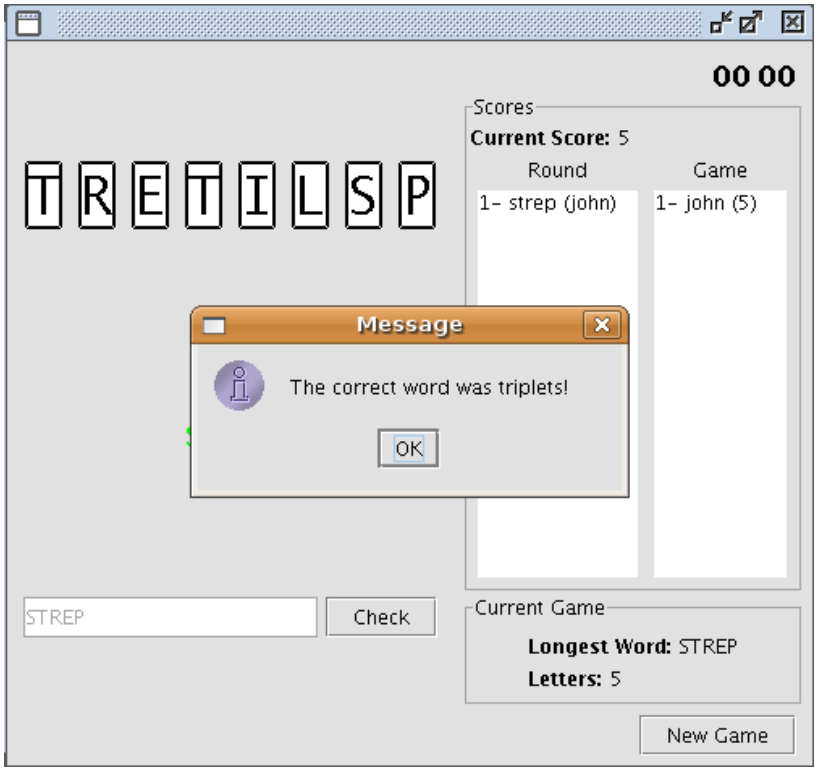

Fig. 9 - Anagram Plug-in: end of game

For starting the game, one of the players must click the “New Game” button. The timer on the upper-right corner will start the countdown. When the time is finished all the players receive a message with the longest word that could be formed with the 8 letters. Score and round information are also updated.

2.2.7 The Collaborative Map Plug-in

Overview

The Collaborative Map Plug-in (CoMa) provides a map display on a Personal Digital Assistant (PDA) connected with a POPEYE laptop via Bluetooth to allow support in orientation based tasks for members of Workspaces especially appropriate for larger outdoor environments.

Fig. 10 - Main map display of the CoMa Plug-in on a PDA

The map display on the PDA as shown in Fig. 10 has different visualisation layers for presentation of different types of information. First of all, the current position of the user is gathered from a position receiver and shown on a geo-referenced map. Furthermore, the current user's position is sent to the other Workspace members to enable them to see the position on their device. The position data is currently based on GNSS (Global Navigation Satellite System) signals. In the future, this positioning mechanism can be replaced or complemented by other systems such as indoor positioning (e.g., via RFID-tags or WLAN signal strength).

Beside the map layer and the user position layer further information types that are displayed are the positions of other members within the user's workspace and specific landmarks or points of interest (POIs) that can be defined and sent by the Workspace members. By this means, members can, for example, define meeting points and show them to others. By clicking on the symbols on the map, the user can get further information about them (e.g., see the profile of a user or a description of a POI) or can send short text messages to other members.

3 Evaluation of the POPEYE Application

During the course of the project several internal and external events were conducted with the goal to evaluate and verify the POPEYE architecture and the POPEYE application. Additionally, tools were realised to support and accompany the POPYE development process.

3.1 Internal evaluation

The most important internal evaluation was done by developer meetings. Several of these developer meetings – each one-week long – took place to ensure the quality of integration and to evaluate the functionality.
The meetings were conducted in different phases of the project to recognise deviations as well as potential problems and risks in the different development stages. During these meetings the integration and functionality of different POPEYE aspects were evaluated informally by the developers.
This procedure gave useful feedback for the further development and supported the preparation of external demonstration events to gain end user feedback.

3.2 Evaluation by demonstration events

Beside the internal evaluation, an important means for the evaluation of the POPEYE applications were the external demonstration events. The objective of the demonstration events was the verification of POPEYE architecture and the verification of the proof-of-concept applications. By invitation of end users and external experts from the domain of collaborative environments, user feedback was gained that gave helpful support in enhancing and consolidating POPEYE integration, usability and functionality. During the project time two demonstration events were accomplished.

The First Demonstration event was held in Paris in October 2007. In addition to the already mentioned objectives, it was also a part of the requirements validation process with regard to the list of requirements defined in the early project phase und documented in Deliverable D2.2 “Description of functional, non functional and technical requirements”. The external experts participating to the demonstration gave valuable feedback for the further development within the POPEYE project. The demonstration itself was based on four different demonstration cases. Each of these cases was intended to highlight different aspects of the POPEYE application such as setup, new users and security. The participants were asked to write down their remarks and comments on the provided feedback forms. These forms also include lists of requirements to evaluate how far the obtained requirements were addressed by the demonstration and application from the users’ point of view. This user feedback was collected after the event and analysed to improve the further development.

The Final demonstration took place at the 6th MINEMA Workshop in Glasgow at Begin of April 2008. This demonstration provides an open, public showcase of POPEYE technologies and follows a demonstration scenario including a complete storyline that covers all major aspects of POPEYE. The workshop participants were asked for comments and feedback to evaluate the final POPEYE application and the project outcome.

Detailed information about the First Demonstration event can be found in the appendix "POPEYE First Demonstrator event proceedings" of Deliverable D8.1 "First POPEYE Demonstration". The Second Demonstration event is described as part of Deliverable D8.2 "Final POPEYE Demonstration".

3.3 The Network Configuration Utility as an evaluation tool

This section deals with the Network Configuration utility developed by the POPEYE team. This tools has a double use: first of all it's a useful tool aiding internal evaluation and validation, and secondly it provides useful visual feedback helping users to understand what's happening "behind the scenes" of the network topology during demonstrations of POPEYE.

3.3.1 Motivation

It is well know by the developer community the difficulty of testing, validating and showing a piece of software. If we add the fact that POPEYE is based on Multi-hop Ad-hoc Networks (MANETS), the problematic highly increases.

One of the main values of the POPEYE framework is the multi-hop communication capability. However, this feature cannot be appreciated when all the devices used during a demonstration are visible to each other, i.e. if the devices are so close physically that the radio waves emitted by any of these devices are received by the rest, we only have a one-hop topology.

Due to the restricted environments used during POPEYE demonstrations (usually a conference room not big enough to have multi-hop topologies), the POPEYE team realised the need to come out with a solution in order to show multi-hop topologies in small spaces.
For that purpose, the team started using MacKill, a tool which allows the filtering of packages coming from the MAC addressed that we have previously defined. That way, it was possible to create the topology we desired by specifying the MACs addresses that each machine has to filter.

At the beginning, these filters had to be manually set. The problem was that this process was a very tedious and prone to error. Furthermore, it was not always easy to show the result to the audience.

Analysing these problems, we realised the necessity of having an easer way of creating multi-hop scenarios. We then came out when another solution, a centralised application allowing to configure the filters of each of the devices and visualising the result graphically. This is what is now called the POPEYE Network Topology Configuration Tool.

3.3.2 Functionality

The configuration tool includes the following functionality:

- A canvas area lets the user configure scenarios with any number of nodes and any links between nodes.
- The topology we can see on the screen, can be physically created at any time (by means of MacKill)
- Physical connectivity can also be checked at any time (ping messages)
- Topologies can be saved or loaded at any time
- Background images can be loaded
- Different mobility models can be used to move nodes around the terrain and change the links accordingly depending on distance between nodes

3.3.3 How it works

The Network Configuration tool follows a Client-Server architecture:

- **MacKillController**: A small application called MacKillController (also distributed with NTCT) acts as the client. This low weight application is responsible for receiving the MAC addresses that must be filtered.

  IMPORTANT: in order to set up certain topology configuration, MacKill must be installed in all the devices taking part of the scenario (including the machine where the GUI is running). The file "mackill.ko" must be located in the following path: /root/popeye/mackill-0.2-kernel-2.6/mackill.ko" and an instance of MacKillController must be running in all the machines

- **Network Topology Configuration Tool**: This is a graphical application and acts as the Server. One instance must be launched in one of the machines.

Next figure depicts a sample scenario. As it can be seen, clients do not need to have direct links with the Server: as long as there exists a path between the Server and all the Clients, any topology can be created.

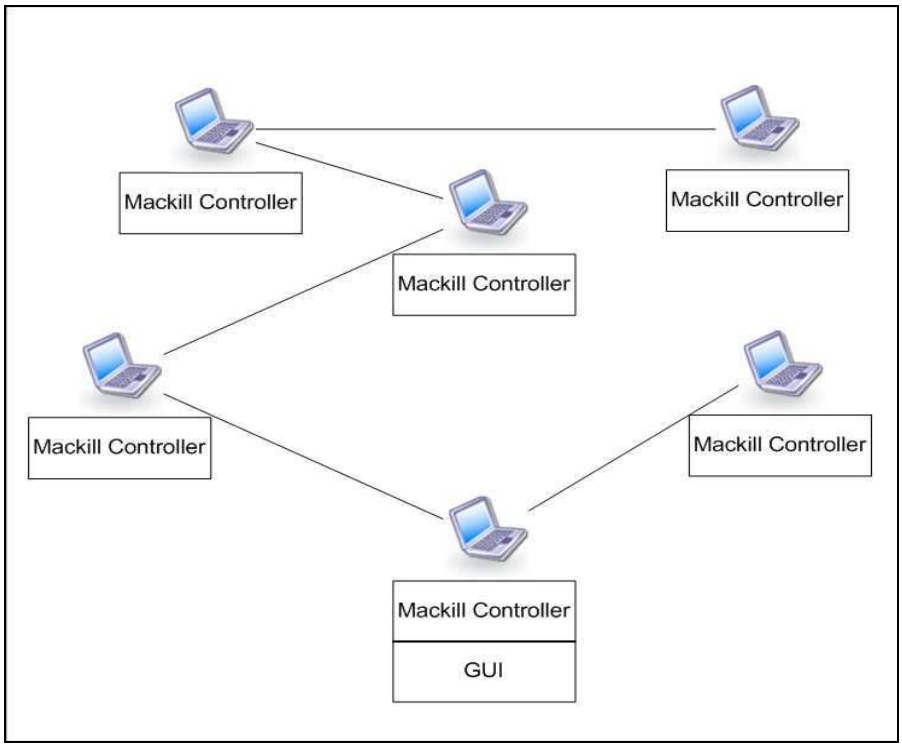

Fig. 11 - Client/Server architecture

3.3.4 User Manual

The graphical user interface is divided into three parts:

- Left: Canvas used to create the network topology
- Right Top: node parameters such as IP and MAC address can be edited here
- Right Button: provide some functionality buttons which let us apply the current topology or check the accessibility between the different nodes

In order to start creating a scenario, the user only needs to click on one of the first two buttons of the left panel (the first will create a node with a Peer icon whereas the second will create a node with a Super Peer icon).
To create links between nodes, click on the centre of the source node and drag to the target node (links are bidirectional).
Topology can be changed at any time by creating or erasing links between nodes.

When a new node is added to the scenario, a new entry on the table located on the right is created. From this table, user can specify the node IP and MAC addresses. Node names can be changed by double clicking on the node image.

The file menu allows three different operations: save the current topology, load a previously saved topology, set up a background image.

Users check the accessibility between nodes by pressing the "CHECK ACCESSIBILITY" button. The application tries to reach all the nodes in the scenario (ping-like messages). At the end of the process, all the peers who where accessible will be displayed in their original colour and the ones not accessible, in red.

The bottom right panel also displays at any moment whether all the nodes in the graph are connected or not. In order to apply certain topology, the graph must be strongly connected. Otherwise, a network partition would be created. In that case, the tool would not be able to reach some of the nodes and therefore, incapable of change their filters.

The "Stop MacKill" button allows the user to stop MacKill in all the nodes involved in the scenario.

The following figures show some examples of possible scenarios.

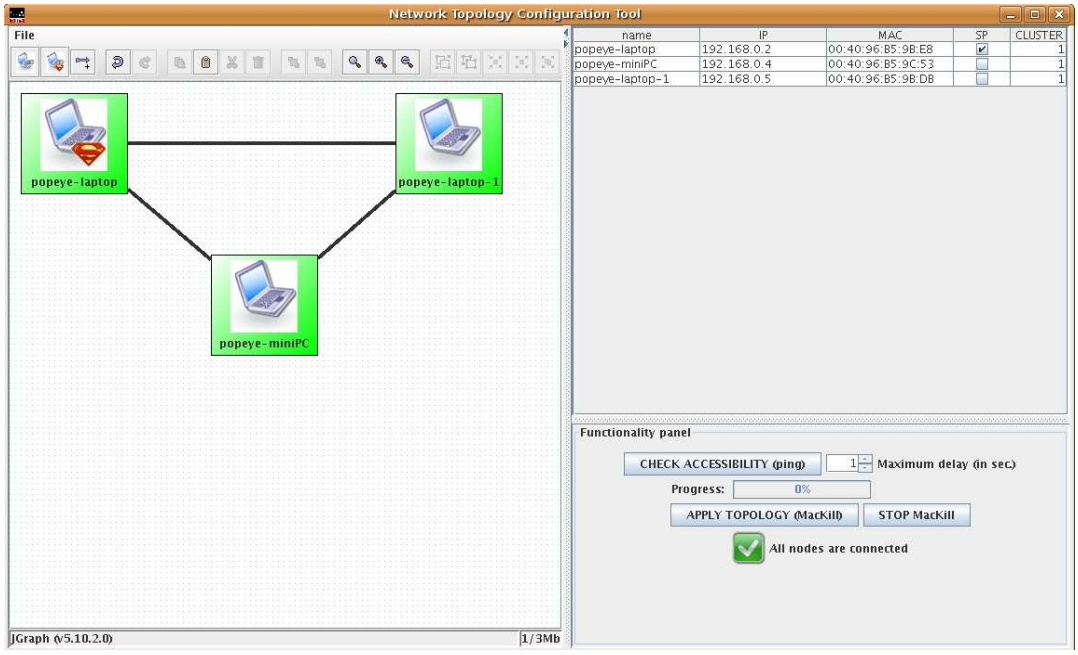

Fig. 12 - Scenario A: 3 nodes distributed in 1 cluster

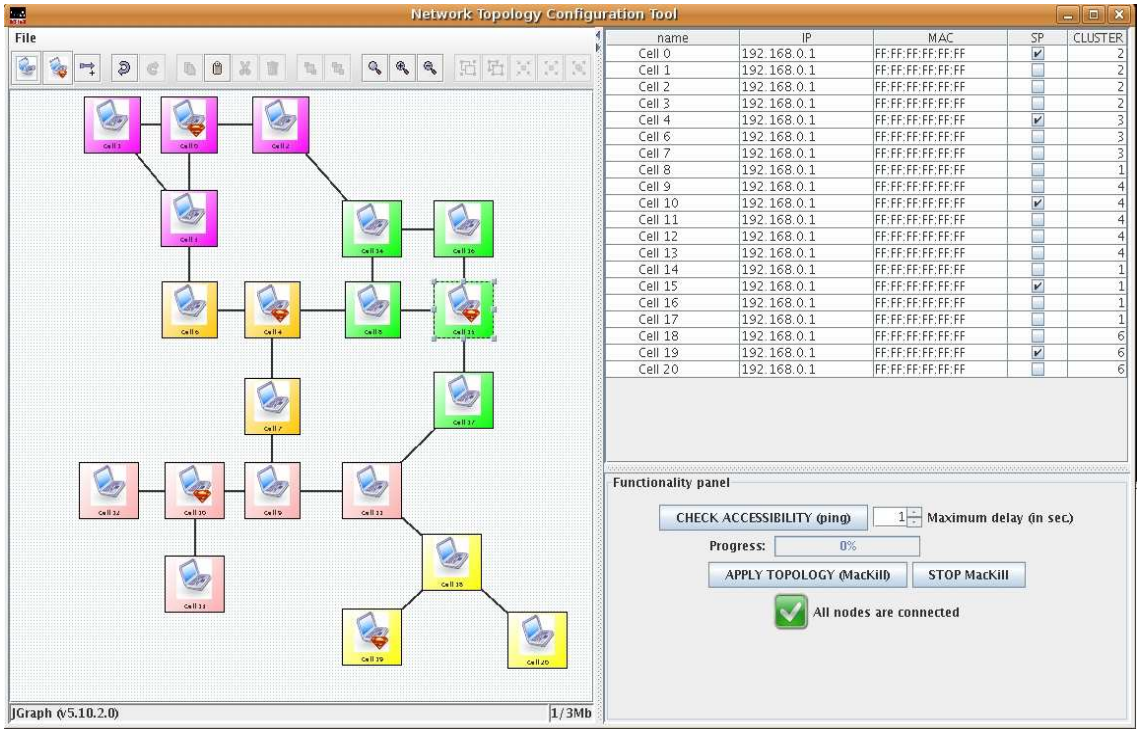

Fig. 13 - Scenario B: 20 nodes distributed in 5 clusters

4 POPEYE Application Development Guidelines

The development of new application functionalities in POPEYE means to construct new Plug-ins for the POPEYE framework. Each Plug-in can then make use of the services of POPEYE, such as communication, security, context, etc…, that are provided by the framework. The remaining of this Section is organised as follows: the POPEYE plug-in-based framework is recalled in Section 4.1 (for further details, please refer to the POPEYE deliverable D5.32); Section 4.2 explains how to write a new plug-in for POPEYE, and finally, Section 4.3 explains how to install plug-ins in POPEYE.

Therefore, Sections 4.1 and 4.2 describe useful information for POPEYE developers, while Section 4.3 describes the steps that a POPEYE user has to perform to extend its own POPEYE with plug-ins already developed and released.

4.1 The POPEYE Plug-in based Framework

The POPEYE Plug-in based architecture is inspired by Charmy[3], a Plug-in based framework developed at the University of L'Aquila. Charmy components and Plug-in infrastructure are intended to be reused and adapted for the POPEYE Plug-in manager. The first service provided by the Plug-in manager is the installation and uninstallation of the Plug-ins. Installation of a plug-in means to create a new instance of a plug-in; in fact it is possible to have active at the same moment more that one instances of the same plug-in (for example more instance of the chat plug-in). Plug-ins installation and uninstallation take into account possible dependencies among plug-ins. In fact some plug-ins can require to have installed other plug-ins. Therefore, a plug-in can be installed only if each plug-in required is already installed. The installation of plug-in is performed by interacting with the POPEYE GUI. This aspect is detailed in Section 4.3.

Finally, the POPEYE plug-in manager provides communication infrastructures for plug-ins. Two kinds of communications are provided: (i) a communication similar to a publish/subscribe mechanism used for instance for plug-ins synchronisation (e.g., a plug-in that has to react to modifications of other plug-ins) and (ii) a communication, similar to components communication, by means of the plug-in interface (i.e., simply invoking plug-in services). The following section will shows example of both these kinds of communication.

[2] POPEYE deliverable D5.3, entitled "POPEYE Core Services Architecture Definition", is publicly available at www.ist-popeye.eu

[3] Charmy Project: Charmy web site. http://www.di.univaq.it/charmy (2004)

4.2 Developing new Plug-ins

From the implementation point of view a POPEYE plug-in is a java class that extends the abstract class `eu.popeye.pluginManager.plugin.Plugin`.
The only one abstract method that the Plugin class has is the method init:

```
public abstract void init();
```

Therefore, a POPEYE plug-in has to implement this method that will be called at the loading of the plug-in. A default stop method is also provided in the abstract class Plugin, but if needed (for example in case of complex data deallocation to be performed or for re-initialisation needed by POPEYE services uses by the plug-in) the method can be overwritten. The default implementation provided by Plugin is as follows:

```
public void stopPlugin(){
  this.pluginManager.getStartedPluginRegistry().remove(this);
}
```

The only one performed operation is the un-registration of the plug-in from the registry of started plug-ins.

Here in the following we show the code needed to write an “Hello World” plug-in of POPEYE. We need only one class that we will call SamplePlugin that extends the Plugin abstract class:

```
package plugin.SamplePlugin;

import java.awt.BorderLayout;

import javax.swing.JTextArea;

import eu.popeye.pluginManager.plugin.Plugin;

public class SamplePlugin extends Plugin{


     public void init() {
          BorderLayout borderLayout = new BorderLayout();
          borderLayout.setHgap(0);

          this.add( new JTextArea("Hello World!!"),
                  BorderLayout.CENTER);
          plugDataWin.getMainPanel().add("Sample Plugin", this);
          this.getParent().setVisible(true);

     }
}
```

The method init in this case contains five lines of code needed to construct the simple graphical elements needed to show in Java a JTextArea with the text “Hello World!!”.
The last two lines of code of this example are the most interesting since they are written to add the plug-in to the POPEYE main panel (the parameters are the name to be shown on the plug-in window and the reference to the plug-in class) and to make it visible, respectively. Fig. 14 shows the Hello World plug-in.

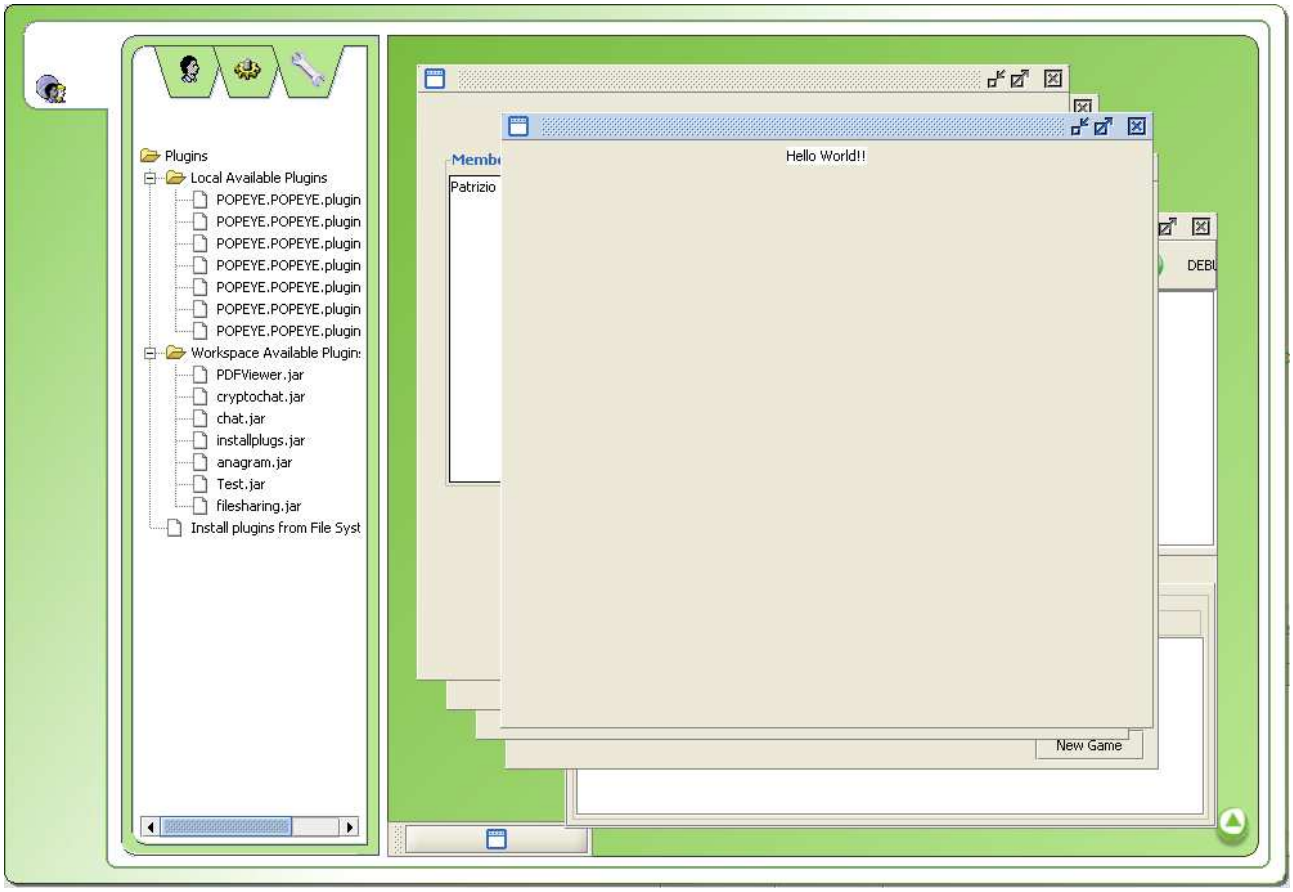

Fig. 14 - Hello World Plug-in

To have a POPEYE plug-in that can be successfully installed in the POPEYE framework we have also to write an XML file containing required information:

```
<?xml version="1.0" encoding="UTF-8"?>
<plugin
  id="POPEYE.POPEYE.plugin.samplePlugin"
  name="SamplePlugin"
  version="1.0.0"
  vendor-name="UDA"
  class="plugin.newPlug.SamplePlugin">

  <runtime>
    <library name="samplePlugin.jar"/>
  </runtime>
</plugin>
```

A plug-in has an ID that is used to univocally refer to the plug-in. A plug-in has also a name, a version, the startup class that is in this case "plugin.newPlug.SamplePlugin". The developer is also forced to generate the jar of the plug-in that will be used to share the plug-in. This jar archive will contain both the .class file of the plug-in and the xml file.

In case of a plug-in, that we will call Requestor, needs to use the services provided by another plug-in, that we will call Provider, then the interface of the provider plug-in must be retrieved and this is performed through the method `getPlugData` of the class `PlugDataManager` that is in its turn retrieved through the `getPlugDataManager` method of the `Plugin` abstract class.

```
plugin.<Provider>.data.PlugData plugData<Provider> =
     (plugin.<Provider>.data.PlugData)
     this.getPlugDataManager().getPlugData("POPEYE.POPEYE.plugin.<
Provider>");
```

Note that we are assuming that the Provider plugin is a real provider, i.e. that it has a PlugData class in the package data used for exporting its interface to other plug-ins. For instance the Hello World plugin is not proving any service to others plug-ins and thus it does not have a PlugData class.
Note also that this is a clear example of dependency between the Requestor and the Provider plug-ins. This dependency must be consistently declared in the plugin.xml file of the Requestor plugin:

```
<requires>
      <import plugin = " POPEYE.POPEYE.plugin.<Provider>"  version="1.0.0"/>
</requires>
```

A plugin can be also kept synchronised with modifications performed on elements of other plug-ins though the subscription of suitable listeners. In the following we show an example assuming that we already retrieved the interface of the Provider plugin as described above:

```
  if(plugData<Provider>!=null)
    plugData<Provider>.get<ElementToSynchr>().addListener(this);
```

4.3 Installing new Plug-ins

To employ applications in POPEYE the implemented plug-ins have to be instantiated. This happens within a Workspace by means of creating a Session that is related to the Workspace and which provides specific configurations and a unique reference to a plugin instance.

The installation of a plugin is supervised by the plugin manager, so that an end user has the ability to search for available plug-ins in the POPEYE environment, to share plug-ins or to receive notifications about shared plug-ins. More specifically, plug-ins that locally reside to the peer are, accordingly to the declared dependencies, loaded at the system startup. After the startup, at each moment a user can start new instances of already installed plug-ins, can install plug-ins that are in the file system (browsing the file system and selecting the jar file of the plugin), or can install plug-ins that are shared by another peer and then that reside on another peer.
The next figure shows the plugin console in the POPEYE framework from which the POPEYE user can install plug-ins. The uninstallation of plug-ins is automatically performed when a plugin is closed.

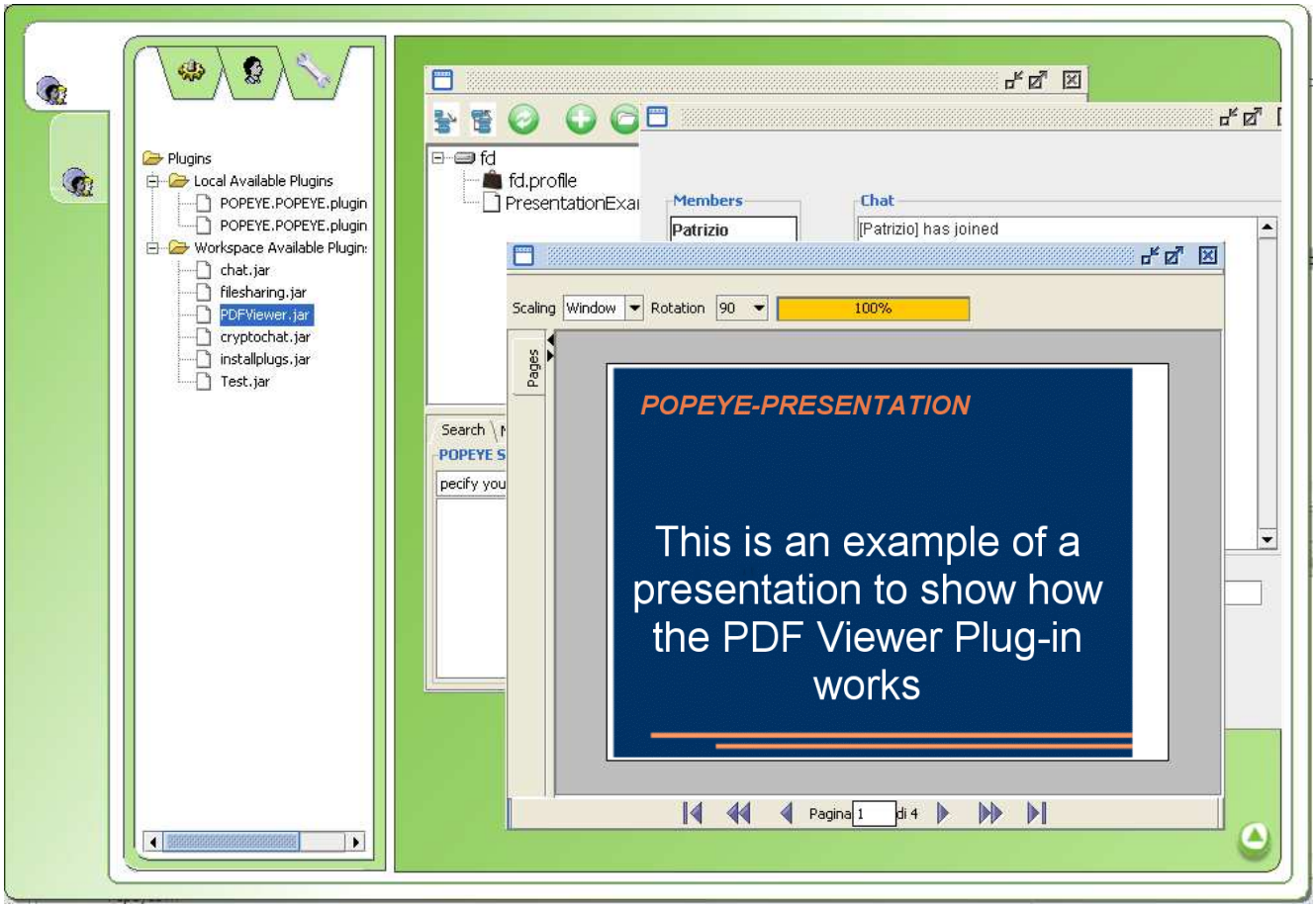

Fig. 15 - Plug-in console

5 Conclusions

The goals of this deliverable were both to provide an evaluation of the current proof-of-concept application built on top of the POPEYE framework, and to provide guidelines for the development of other applications on top of the same framework through the use of the flexible Plug-in based architecture supplied by POPEYE.

The current set of developed Plug-ins is described in the first part of the document. These were the ones developed specifically for the baseline conference scenario used as reference for POPEYE proof-of-concept development but many of them are quite general-purpose. Moreover, a tool which though not part of POPEYE software, but which turned up to be very useful for the evaluation of this is also described in this document, the Network Configuration Utility.

Developing new POPEYE applications is made easy and straightforward by the comprehensive and flexible Plug-in infrastructure on which the POPEYE framework is based. The technique for developing new Plug-ins and installing them on top of POPEYE application is described in the second part of this document.

We hope the present public guidelines and instructions effectively facilitates the development of mobile P2P e-Collaboration application on top of POPEYE framework. Further, developers are invited to refer to the POPEYE web site at www.ist-popeye.eu for more publicly available documentation, and may even join the POPEYE user community.

ANNEX A: Popeye Source Code

The POPEYE source code is made publicly available as open source on SourceForge at the address:
http://sourceforge.net/projects/popeye-cwe/

Also the ManetConfig tool has been made publicly available on SourceForge, at the address:
http://sourceforge.net/projects/manetconfig/

The POPEYE source code and the ManetConfig tool source code are also provided part of D6.3 on CD Rom as is in June 2008.